Orchestrating a brighter world

NEC

SR-IOV ixgbe driver limitations and improvement

2016-07-15

Hiroshi Shimamoto

Orchestrating a brighter world

NEC brings together and integrates technology and expertise to create the ICT-enabled society of tomorrow.
We collaborate closely with partners and customers around the world, orchestrating each project to ensure all its parts are fine-tuned to local needs.

Every day, our innovative solutions for society contribute to greater safety, security, efficiency and equality, and enable people to live brighter lives.

Hiroshi Shimamoto

- Working for NEC

- Main activities in open source community
  - Carrier Grade Linux
  - Virtualization
  - Packet Processing, DPDK

to Make Linux system usable for Telecom Carriers

Outline

SR-IOV and ixgbe implementation

SR-IOV ixgbe limitations for NFV

Addressing the issues

Future work and possible security issues

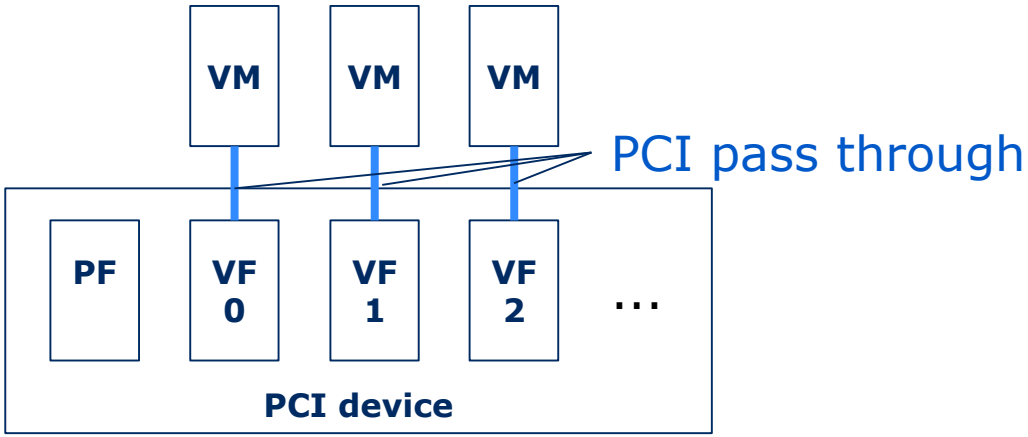
SR-IOV

What is SR-IOV

- One of device virtualization technology
- Most of people here may know it

In SR-IOV enabled PCI device provides PF (Physical Function) and VF (Virtual Function)

- VF will be used directly in VM



Why SR-IOV?

- I/O in virtual machines is a performance bottleneck
- Especially packet switching between VM and host

- Technologies were introduced to address network performance
  - PCI pass through
  - SR-IOV
  - DPDK

SR-IOV and etc. comparison

Brief pros and cons each technology

	PCI Pass through	SR-IOV	DPDK
pros	Hardware performance	Multiple VMs Hardware performance	Multiple VMs Software flexibility No special driver in guest
cons	Single VM	Device support VF driver in guest	Consume much CPU power

How SR-IOV implemented in Intel 82599

- Target device: Intel 82599 10GbE Controller and ixgbe driver for PF and ixgbevf driver for VF

- There are 64 VMDq (Virtual Machine Device queue) in 82599
  - Calls this queue pool
  - For SR-IOV, each VF has associated pool
  - 82599 switches a packet to pool (VF)

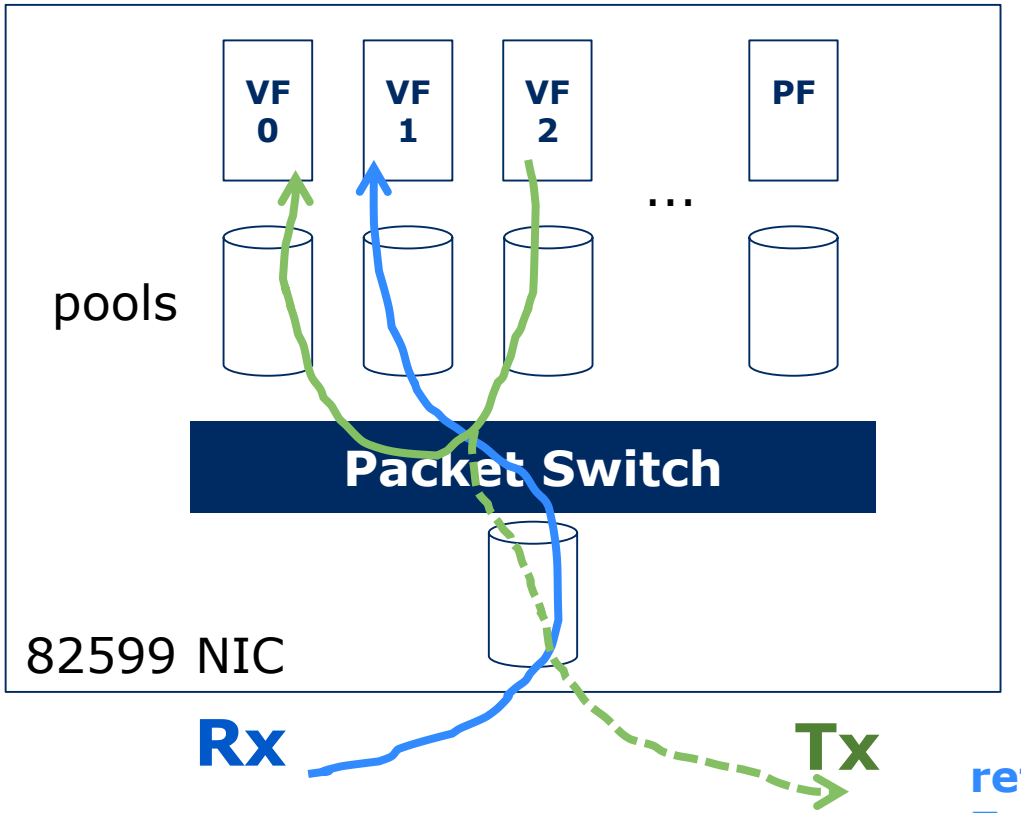
- The current ixgbe driver map

VF	pool
0	0
1	1
:	:
N	N
PF	N+1

Intel 82599 10GbE controller datasheet URL
http://www.intel.com/content/www/us/en/embedded/products/networking/82599-10-gbe-controller-datasheet.html

Packet switching in Intel 82599

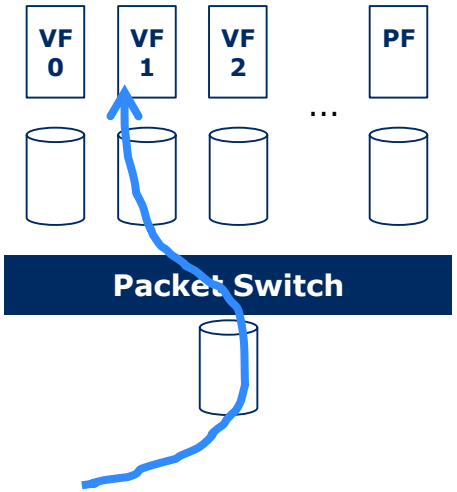
- In SR-IOV mode
- 82599 chip switches a packet to corresponding pool



ref. datasheet
7.10.3 Packet Switching

Rx packet switching

Receive a packet from outside



Step

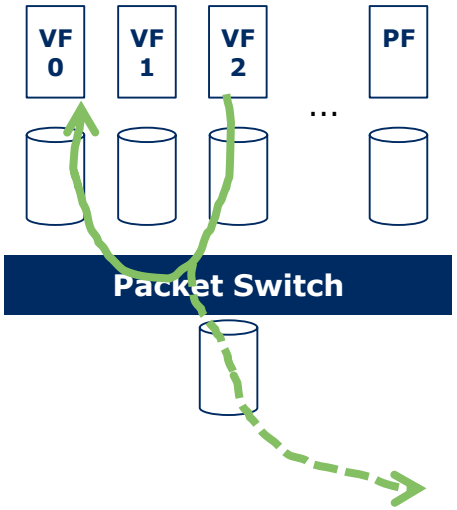
1. Exact unicast or multicast match
2. Broadcast
3. Unicast hash*
4. Multicast hash
5. Multicast promiscuous*
6. VLAN group
7. Default pool
8. Ethertype filters
9. PFVFRE
10. Mirroring*

Note*
driver didn't support these features

ref. datasheet
7.10.3.3 Rx Packet Switching

Tx packet switching

Receive a packet from device (PF and/or VF)



Step

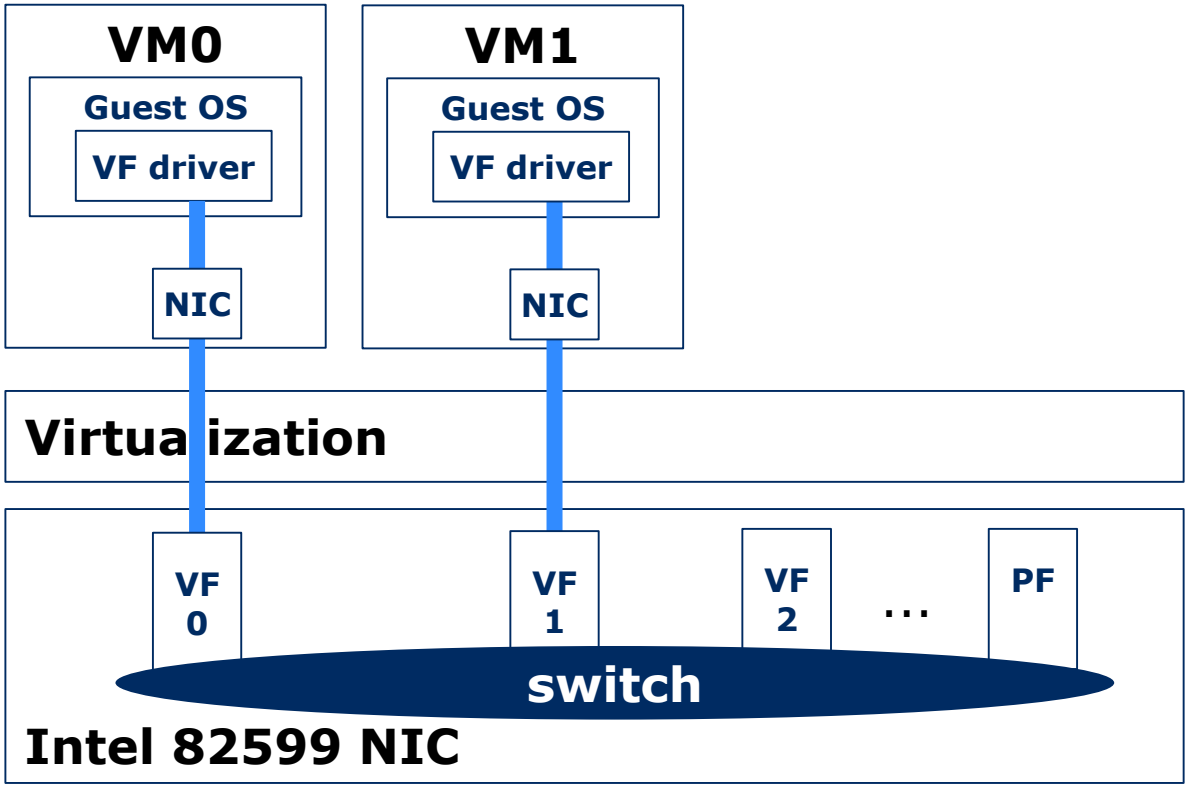
1. Exact unicast or multicast match
2. Broadcast
3. Unicast hash*
4. Multicast hash
5. Multicast promiscuous*
6. Filer source pool
7. VLAN groups
8. Forwarding to the network
9. PFVFRE
10. Mirroring*
11. PFVFRE

Note*
driver didn’t support these features

ref. datasheet
7.10.3.4 Tx Packet Switching

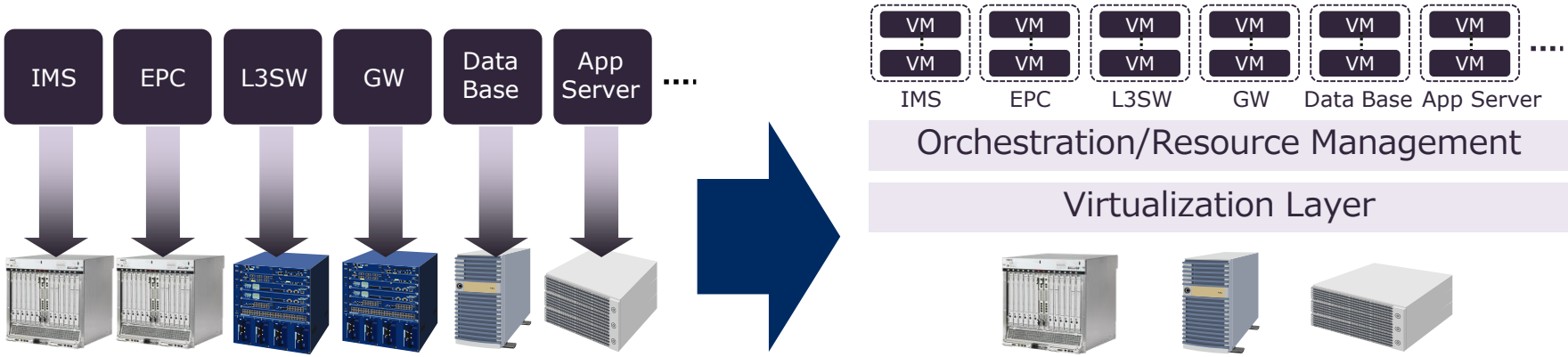
Use Intel 82599 NIC SR-IOV as switch

Typical use case



NFV (Network Function Virtualization)

- Using virtualization technology, implements Network Functions on general purpose hardware, to improve flexibility, agility and efficiency



Dedicated/Specific Hardware

General Purpose Hardware

- Network Function is virtualized

→ VNF (Virtual Network Function) is realized on VM

SR-IOV ixgbe driver limitations for NFV

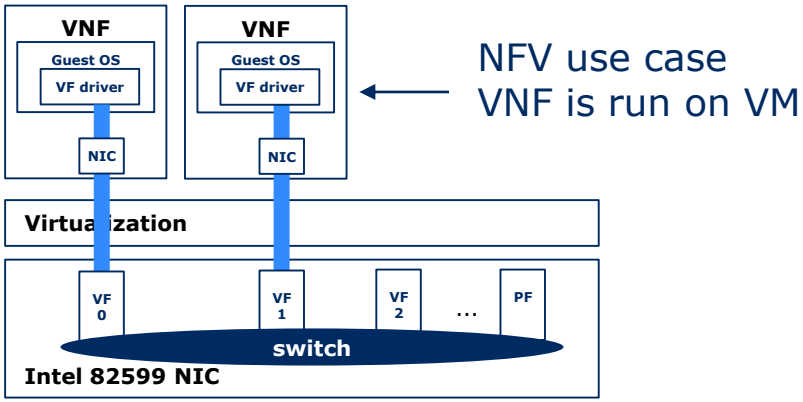
Using Intel 82599 and SR-IOV, 3 critical limitations for NFV

- ✓ **VLAN filtering**
- ✓ **Multicast addresses**
- ✓ **Unicast promiscuous**

Come from hardware limitation and software (driver) limitation

Explain with 2 use cases

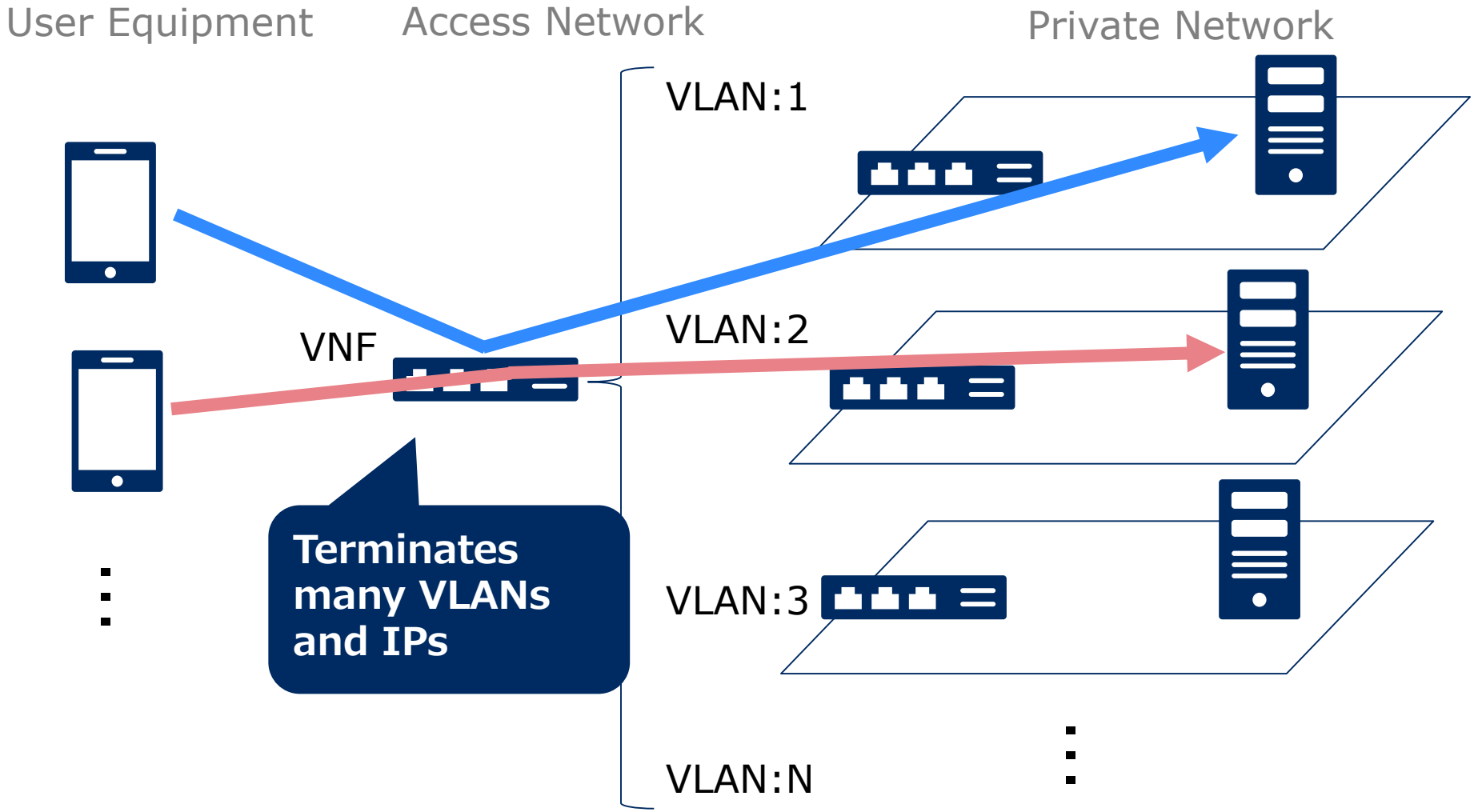
- Router
- Layer 2 switch



NFV use case 1

Network Function: Router

- VLAN is used to virtualize the network



VLAN filtering limitation

- Intel 82599 has hardware VLAN filter

- The problem comes from
  - ✓ VLAN filter has only 64 entries
  - ✓ Driver always enable VLAN filter if SR-IOV enabled

→ Only 64 VLANs can be used with SR-IOV

Example: VF device returns error against adding VLAN

Try to create 64 VLANs on VM

```
# for i in `seq 1 64`; do
> echo "vlan $i"
> ip link add link ens6 name ens6.$i type vlan id $i
> done

vlan 1
vlan 2
  :
vlan 63
vlan 64
RTNETLINK answers: Permission denied
```

Note
The ixgbe driver use the first entry for VLAN 0, that means actual the number of VLANs is 63

Enabling VLAN filter automatically in driver (latest)

upstream

```
static void ixgbe_vlan_promisc_enable(struct ixgbe_adapter *adapter)
{
    :
        switch (hw->mac.type) {
        case ixgbe_mac_82599EB:
        case ixgbe_mac_X540:
        case ixgbe_mac_X550:
        case ixgbe_mac_X550EM_x:
        case ixgbe_mac_x550em_a:
        default:
                if (adapter->flags & IXGBE_FLAG_VMDQ_ENABLED)
                        break;
                /* fall through */
        case ixgbe_mac_82598EB:
                /* legacy case, we can just disable VLAN filtering */
                vlnctrl = IXGBE_READ_REG(hw, IXGBE_VLNCTRL);
                vlnctrl &= ~(IXGBE_VLNCTRL_VFE | IXGBE_VLNCTRL_CFIEN);
                IXGBE_WRITE_REG(hw, IXGBE_VLNCTRL, vlnctrl);
                return;
        }
```

Enabling VLAN filter automatically in driver (old)

previous version

```
void ixgbe_set_rx_mode(struct net_device *netdev)
{
    :
    :
        vlnctrl &= ~(IXGBE_VLNCTRL_VFE | IXGBE_VLNCTRL_CFIEN);
        if (netdev->flags & IFF_PROMISC) {
                hw->addr_ctrl.user_set_promisc = true;
                fctrl |= (IXGBE_FCTRL_UPE | IXGBE_FCTRL_MPE);
                vmolr |= IXGBE_VMOLR_MPE;
                /* Only disable hardware filter vlans in promiscuous mode
                 * if SR-IOV and VMDQ are disabled - otherwise ensure
                 * that hardware VLAN filters remain enabled.
                 */
                if (adapter->flags & (IXGBE_FLAG_VMDQ_ENABLED |
                                      IXGBE_FLAG_SRIOV_ENABLED))
                        vlnctrl |= (IXGBE_VLNCTRL_VFE | IXGBE_VLNCTRL_CFIEN);
```

Multicast address limitation

- The interface (PF-VF mailbox API) limits the number of addresses
- Only first 30 multicast addresses can be registered
- Overflowed addresses are silently dropped

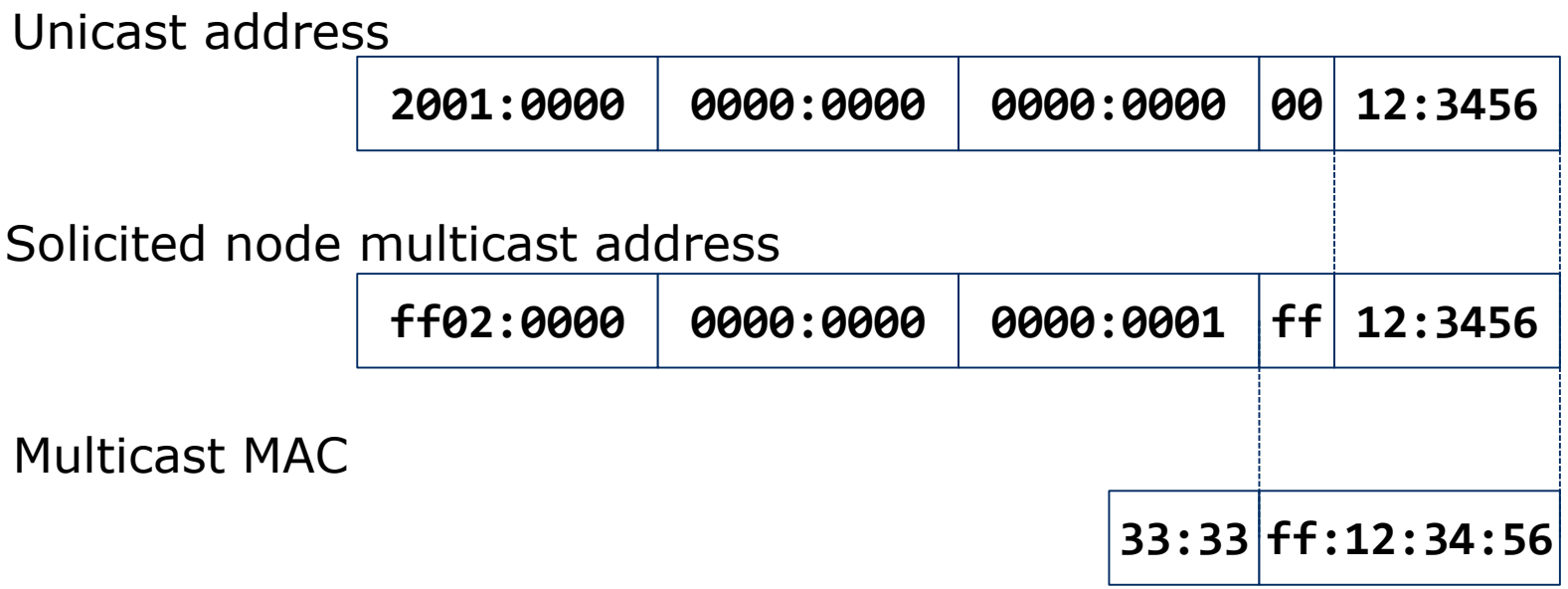
```
/* Each entry in the list uses 1 16 bit word.  We have 30
 * 16 bit words available in our HW msg buffer (minus 1 for the
 * msg type).  That's 30 hash values if we pack 'em right.  If
 * there are more than 30 MC addresses to add then punt the
 * extras for now and then add code to handle more than 30 later.
 * It would be unusual for a server to request that many multi-cast
 * addresses except for in large enterprise network environments.
 */

cnt = netdev_mc_count(netdev);
if (cnt > 30)
        cnt = 30;
msgbuf[0] = IXGBE_VF_SET_MULTICAST;
msgbuf[0] |= cnt << IXGBE_VT_MSGINFO_SHIFT;
```

→ Having Multicast promiscuous is a solution

Why so many multicast addresses?

Our case is to support many IPv6 addresses on VF

- For Neighbor Discovery, each IPv6 address requires corresponding multicast address



Example: Cannot receive multicast packet

- Some multicast packets are not passed to VF, then failed to ND

- Assign 30 IPv6 addresses on VM (different network address)

```
# for i in `seq 1 30`; do
> ip -6 addr add 2001:$i::1:$i/64 dev ens6
> done
```

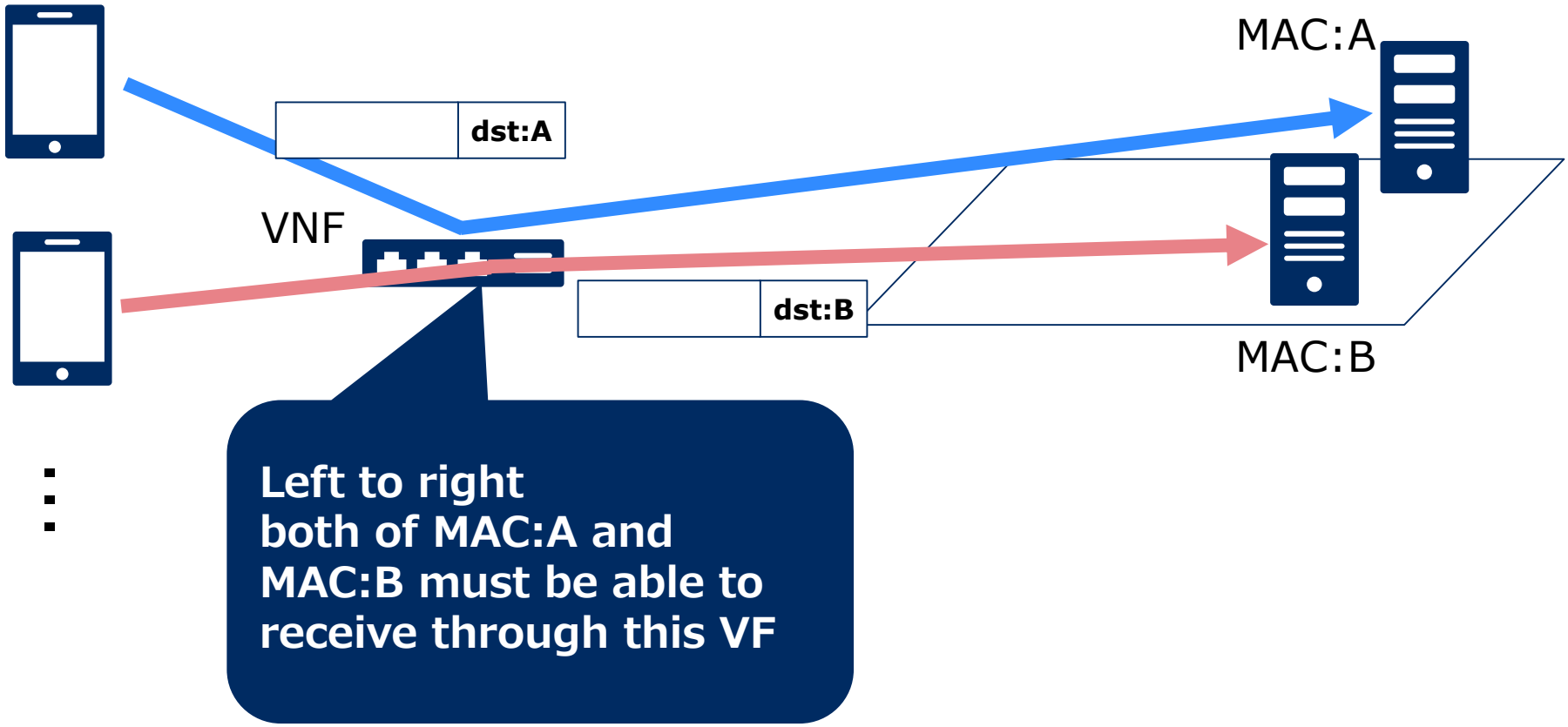
- Ping from other physical machine to VM

```
# for i in `seq 1 30`; do ping6 -w 1 -c 1 -I eth2 2001:$i::1:$i; done
    :
PING 2001:28::1:28(2001:28::1:28) from 2001:28::1 eth2: 56 data bytes

--- 2001:28::1:28 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1000ms
```

NFV use case 2

Network Function: L2 switch

User Equipment Access Network Private Network



Unicast promiscuous

- Only packets which match the registered MAC can be received

- VNF should be able to handle every MAC addresses in network
- Hard to register all MAC addresses in L2 network

→ We want Unicast promiscuous feature in VF

SR-IOV limitations for NFV and current status

VLAN filtering

- Only 64 VLANs can be used

→ Proposed to add an option to disable hardware VLAN filter, but not accepted

Multicast addresses

- Only 30 Multicast addresses can be used

→ Implemented VF Multicast promiscuous mode in ixgbe/ixgbevf driver in Linux 4.4

Unicast promiscuous

- Single unicast MAC address can be used

→ Hardware limitation

Addressing Multicast addresses limitation

- There is a hardware feature in 82599
  - VF Multicast promiscuous mode
- But driver didn’t support that feature
- Implement new PF-VF mailbox API in ixgbe and ixgbevf
- First, automatically enable to VF multicast promiscuous when the number of addresses overs 30
- Suggested way, implement xcast mode in VF
  - There is ALLMULTI flag that means that every multicast packet is received in this device

→ Accepted in Linux 4.4

PF-VF mailbox APIs

There are mailbox APIs

- Communicate between PF and VF

version	API	description
legacy	RESET	VF requests reset
	SET_MAC_ADDR	VF requests PF to set MAC addr
	SET_MULTICAST	VF requests PF to set MC addr
	SET_VLAN	VF requests PF to set VLAN
1.0	SET_LPE	VF requests PF to set VMOLR.LPE
	SET_MACVLAN	VF requests PF for unicast filter
	API_NEGOTIATE	negotiate API version
1.1	GET_QUEUES	get queue configuration
1.2	GET_RETA	VF requests for RETA
	GET_RSS_KEY	get RSS key
	UPDATE_XCAST_MODE	**VF requests PF to set MC mode**

Implementation (PF ixgbe)

Handle UPDATE_XCAST_MODE API

```
--- a/drivers/net/ethernet/intel/ixgbe/ixgbe_sriov.c
+++ b/drivers/net/ethernet/intel/ixgbe/ixgbe_sriov.c

    :

@@ -1066,6 +1122,9 @@ static int ixgbe_rcv_msg_from_vf(struct ixgbe_adapter
*adapter, u32 vf)
        case IXGBE_VF_GET_RSS_KEY:
                retval = ixgbe_get_vf_rss_key(adapter, msgbuf, vf);
                break;
+       case IXGBE_VF_UPDATE_XCAST_MODE:
+               retval = ixgbe_update_vf_xcast_mode(adapter, msgbuf, vf);
+               break;
        default:
                e_err(drv, "Unhandled Msg %8.8x¥n", msgbuf[0]);
                retval = IXGBE_ERR_MBX;
```

Implementation (VF ixgbevf)

Request to PF (from ixgbevf_set_rx_mode)

```
--- a/drivers/net/ethernet/intel/ixgbevf/ixgbevf_main.c
+++ b/drivers/net/ethernet/intel/ixgbevf/ixgbevf_main.c
@@ -1894,9 +1894,17 @@ static void ixgbevf_set_rx_mode(struct net_device
*netdev)
 {
        struct ixgbevf_adapter *adapter = netdev_priv(netdev);
        struct ixgbe_hw *hw = &adapter->hw;
+       unsigned int flags = netdev->flags;
+       int xcast_mode;
+
+       xcast_mode = (flags & IFF_ALLMULTI) ? IXGBEVF_XCAST_MODE_ALLMULTI :
+                    (flags & (IFF_BROADCAST | IFF_MULTICAST)) ?
+                    IXGBEVF_XCAST_MODE_MULTI : IXGBEVF_XCAST_MODE_NONE;

        spin_lock_bh(&adapter->mbx_lock);

+       hw->mac.ops.update_xcast_mode(hw, netdev, xcast_mode);
+
        /* reprogram multicast list */
        hw->mac.ops.update_mc_addr_list(hw, netdev);
```

Security consideration

Enabling VF Multicast promiscuous mode causes security issues

- ✓ Can see all multicast packets through this device
- ✓ Can hurt performance

  NIC duplicates packets and does DMA to each pool

→ Make only **trusted VF** can enable Multicast promiscuous mode

```
static int ixgbe_update_vf_xcast_mode(struct ixgbe_adapter *adapter,
                                      u32 *msgbuf, u32 vf)
{
    :

        if (xcast_mode > IXGBEVF_XCAST_MODE_MULTI &&
            !adapter->vfinfo[vf].trusted) {
                xcast_mode = IXGBEVF_XCAST_MODE_MULTI;
        }
```

Implement functionality to trust VF

- Add new operation “set_vf_trust” in net_device_ops
- Also add support VF trust operation in iproute2 (ip command)

```
# ip link set dev enp3s0f0 vf 1 trust on
(dmesg)
kernel: ixgbe 0000:03:00.0 enp3s0f0: VF 1 is trusted
kernel: ixgbevf 0000:03:10.2: NIC Link is Down
kernel: ixgbe 0000:03:00.0 enp3s0f0: VF Reset msg received from vf 1
kernel: ixgbevf 0000:03:10.2: NIC Link is Up 10 Gbps

# ip link set dev enp3s0f0 vf 1 trust off
(dmesg)
kernel: ixgbe 0000:03:00.0 enp3s0f0: VF 1 is not trusted
kernel: ixgbevf 0000:03:10.2: NIC Link is Down
kernel: ixgbe 0000:03:00.0 enp3s0f0: VF Reset msg received from vf 1
kernel: ixgbevf 0000:03:10.2: NIC Link is Up 10 Gbps
```

Note
When trusted state is changed, target vf is reset

Future work and possible security issues

Still, there are limitations

- ✓ **VLAN filtering**
- ✓ **Unicast promiscuous**

Possible security issues

- ✓ VLAN filter is not isolated
- ✓ Multicast hash table is not handled strictly

VLAN filtering

- Disabling hardware VLAN filtering may solve this issue
- But
  - ✓ It could break existing network feature (DCB, FCoE)
  - ✓ Broadcast(Multicast) storm could cause performance degradation
  - ✓ Security issue, BC/MC packets can be seen in every VF
- Maybe okay if the network and VMs are well managed
- Another point is that there is no suitable knob to do it now
  - What command is right to turn VLAN filter off in general

Unicast promiscuous

- Supporting this feature in hardware is the best
- No VF unicast promiscuous feature in 82599 unfortunately

Hardware support in X540 and X550

- Later NIC chips, X540 and X550 support VLAN promiscuous and Unicast promiscuous mode per VF
- The issues could be solved with X540/X550 chip

- To make framework for X540/X550 and use the same semantics for 82599 may be needed

Field		Bit(s)		Description
Reserved		23:0	**21:0**	Reserved
	UPE		**22**	**Unicast Promiscuous Enable**
	VPE		**23**	**VLAN Promiscuous Enable**
AUPE		24		Accept Untagged Packet Enable
ROMPE		25		Receive Overflow Multicast Packets
ROPE		26		Receive MAC Filters Overflow
BAM		27		Broadcast Accept
MPE		28		Multicast Promiscuous Enable
Reserved		31:29		Reserved

new feature bit

PF VM L2Control Register

Ideas

- Mirroring
- Unicast hash

Those features are not used/supported in driver

Possible security issues

In the current ixgbe/ixgbevf implementation, there are 2 issues to be considered

- ✓ VLAN filter is not isolated
  - Single VLAN filter table
  - No limitation to request new VLAN from VF
  - If a VM requests 64 VLANs, other VMs never use different VLANs
- ✓ Single multicast hash table
  - Using multicast hash table for switching multicast packet to VF
  - SET_MULTICAST API is only for setting hash value, no unset functionality
  - Manipulating IP address assignment can make VF to have multicast promiscuous behavior

VLAN filter is not isolated

- If a VM uses all VLANs, other VM can’t make new VLAN

```
(VM0) # for i in `seq 1 64`; do
> echo "vlan $i"
> ip link add link ens6 name ens6.$i type vlan id $i
> done

vlan 1
vlan 2
  :
vlan 63
vlan 64
RTNETLINK answers: Permission denied

(VM1) # ip link add link ens6 name ens6.64 type vlan id 64
RTNETLINK answers: Permission denied
```

and if VM does not shutdown gracefully,
registered VLAN filter entry remains

Single multicast hash table

- ixgbe driver uses multicast hash table (MTA register)

- Multicast Table Array is a 4Kb bitmap
  i. 82599 make 12 bits hash value from MAC
  ii. If corresponding bit in MTA is set, it means hit
  iii. Check VF capability PFVFL2FLT.ROMPE and transfer packet

→ If every bits in MTA are set, every multicast packet matches and transferred to pool

- MTA bit is set by SET_MULTICAST API and no unset API
- PFVFL2FLT.ROMPE bit always set by receiving SET_MULTICAST API message

- Long living host may have unnecessary bits in MTA

Example: Can receive all multicast packet

Invoke SET_MULTICAST from VF each IP address

```
# for i in `seq 1 30`; do
> ip -6 addr add 2001:$i::1:$i/64 dev ens6
> ip -6 addr del 2001:$i::1:$i/64 dev ens6
> done
```

Assign 30 IP addresses again

```
# for i in `seq 1 30`; do
> ip -6 addr add 2001:$i::1:$i/64 dev ens6
> done
```

Ping from other physical machine to VM

```
# for i in `seq 1 30`; do ping6 -w 1 -c 1 -I eth2 2001:$i::1:$i; done
  :
PING 2001:30::1:30(2001:30::1:30) from 2001:30::1 eth2: 56 data bytes
64 bytes from 2001:30::1:30: icmp_seq=1 ttl=64 time=0.947 ms

--- 2001:30::1:30 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.947/0.947/0.947/0.000 ms
```

Summary

- SR-IOV in Intel 82599 and ixgbe driver implementation
- SR-IOV ixgbe driver limitations for NFV
  - VLAN filtering
  - Multicast addresses
  - Unicast promiscuous
- Addressing Multicast addresses issue
  - Implement new mailbox API and ndo VF trust
- Future work and possible security issues
- Questions?
- E-Mail: h-shimamoto@ct.jp.nec.com


Orchestrating a brighter world
NEC